Foundations of Computer Science

Lazy lists: higher-order and numeric computations

Dr. Robert Harle & Dr. Jeremy Yallop

2020–2021

filtering

```
In[1]: let rec filterq p = function
         | Nil -> Nil
         | Cons(x, xf) ->
           if p x then
            Cons (x, fun () -> filterq p (xf ()))
           else
            filterq p (xf ())

Out[1]: val filterq : ('a -> bool) -> 'a seq -> 'a seq = <fun>
```

The infinite sequence x, $f(x)$, $f(f(x))$, ...

```
In[2]: let rec iterates f x = Cons (x, fun () -> iterates f (f x))

Out[2]: val iterates : ('a -> 'a) -> 'a -> 'a seq = <fun>
```

filtering

In[1]:
```
let rec filterq p = function
| Nil -> Nil
| Cons(x, xf) ->
    if p x then
     Cons (x, fun () -> filterq p (xf ()))
    else
     filterq p (xf ())
```

Out[1]:
```
val filterq : ('a -> bool) -> 'a seq -> 'a seq = <fun>
```

The infinite sequence x, $f(x)$, $f(f(x))$, ...

In[2]:
```
let rec iterates f x = Cons (x, fun () -> iterates f (f x))
```

Out[2]:
```
val iterates : ('a -> 'a) -> 'a -> 'a seq = <fun>
```

filtering

```
In[1]: let rec filterq p = function
       | Nil -> Nil
       | Cons(x, xf) ->
          if p x then
           Cons (x, fun () -> filterq p (xf ()))
          else
           filterq p (xf ())
Out[1]: val filterq : ('a -> bool) -> 'a seq -> 'a seq = <fun>
```

The infinite sequence x, $f(x)$, $f(f(x))$, ...

```
In[2]: let rec iterates f x = Cons (x, fun () -> iterates f (f x))
Out[2]: val iterates : ('a -> 'a) -> 'a -> 'a seq = <fun>
```

filtering

```
In[1]: let rec filterq p = function
       | Nil -> Nil
       | Cons(x, xf) ->
          if p x then
           Cons (x, fun () -> filterq p (xf ()))
          else
           filterq p (xf ())
Out[1]: val filterq : ('a -> bool) -> 'a seq -> 'a seq = <fun>
```

The infinite sequence x, $f(x)$, $f(f(x))$, ...

```
In[2]: let rec iterates f x = Cons (x, fun () -> iterates f (f x))
Out[2]: val iterates : ('a -> 'a) -> 'a -> 'a seq = <fun>
```

filtering

```
In[1]: let rec filterq p = function
       | Nil -> Nil
       | Cons(x, xf) ->
          if p x then
           Cons (x, fun () -> filterq p (xf ()))
          else
           filterq p (xf ())
Out[1]: val filterq : ('a -> bool) -> 'a seq -> 'a seq = <fun>
```

The infinite sequence x, $f(x)$, $f(f(x))$, ...

```
In[2]: let rec iterates f x = Cons (x, fun () -> iterates f (f x))
Out[2]: val iterates : ('a -> 'a) -> 'a -> 'a seq = <fun>
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

Example:

```
val filterq : ('a -> bool) -> 'a seq -> 'a seq
val iterates : ('a -> 'a) -> 'a -> 'a seq
```

```
 In[3]: let myseq = iterates (fun x -> x + 1) 1
Out[3]: val myseq : int seq = Cons (1, <fun>)
 In[4]: filterq (fun x -> x = 1) myseq
Out[4]: - : int seq = Cons (1, <fun>)
 In[5]: filterq (fun x -> x = 100) myseq
Out[5]: - : int seq = Cons (100, <fun>)
 In[6]: filterq (fun x -> x = 0) myseq
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
In[7]:  succ 1

Out[7]: - : 2 = int

In[8]:  let myseq = iterates succ 1

Out[8]: val myseq : int seq = Cons (1, <fun>)

In[9]:  (=) 1 2

Out[9]: - : bool = false
```

Use = function, partially applied:

```
In[10]:  filterq ((=) 100) myseq

Out[10]: - : int seq = Cons (100, <fun>)

In[11]:  filterq ((=) 0) myseq
           ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
In[7]:  succ 1
Out[7]: - : 2 = int
In[8]:  let myseq = iterates succ 1
Out[8]: val myseq : int seq = Cons (1, <fun>)
In[9]:  (=) 1 2
Out[9]: - : bool = false
```

Use = function, partially applied:

```
In[10]:  filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
In[11]:  filterq ((=) 0) myseq
           ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
 In[7]: succ 1
Out[7]: - : 2 = int
 In[8]: let myseq = iterates succ 1
Out[8]: val myseq : int seq = Cons (1, <fun>)
 In[9]: (=) 1 2
Out[9]: - : bool = false
```

Use = function, partially applied:

```
In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
In[11]: filterq ((=) 0) myseq
          ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
  In[7]: succ 1
 Out[7]: - : 2 = int
  In[8]: let myseq = iterates succ 1
 Out[8]: val myseq : int seq = Cons (1, <fun>)
  In[9]: (=) 1 2
 Out[9]: - : bool = false
```

Use = function, partially applied:

```
 In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
 In[11]: filterq ((=) 0) myseq
           ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
 In[7]: succ 1
Out[7]: - : 2 = int
 In[8]: let myseq = iterates succ 1
Out[8]: val myseq : int seq = Cons (1, <fun>)
 In[9]: (=) 1 2
Out[9]: - : bool = false
```

Use = function, partially applied:

```
In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
In[11]: filterq ((=) 0) myseq
          ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
 In[7]: succ 1
Out[7]: - : 2 = int
 In[8]: let myseq = iterates succ 1
Out[8]: val myseq : int seq = Cons (1, <fun>)
 In[9]: (=) 1 2
Out[9]: - : bool = false
```

Use = function, partially applied:

```
In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
In[11]: filterq ((=) 0) myseq
          ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
 In[7]: succ 1
Out[7]: - : 2 = int
 In[8]: let myseq = iterates succ 1
Out[8]: val myseq : int seq = Cons (1, <fun>)
 In[9]: (=) 1 2
Out[9]: - : bool = false
```

Use = function, partially applied:

```
In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
In[11]: filterq ((=) 0) myseq
          ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
 In[7]: succ 1

Out[7]: - : 2 = int

 In[8]: let myseq = iterates succ 1

Out[8]: val myseq : int seq = Cons (1, <fun>)

 In[9]: (=) 1 2

Out[9]: - : bool = false
```

Use = function, partially applied:

```
 In[10]: filterq ((=) 100) myseq

Out[10]: - : int seq = Cons (100, <fun>)

 In[11]: filterq ((=) 0) myseq

         ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
  In[7]: succ 1
 Out[7]: - : 2 = int
  In[8]: let myseq = iterates succ 1
 Out[8]: val myseq : int seq = Cons (1, <fun>)
  In[9]: (=) 1 2
 Out[9]: - : bool = false
```

Use = function, partially applied:

```
 In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
 In[11]: filterq ((=) 0) myseq
           ...
```

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
 In[7]: succ 1
Out[7]: - : 2 = int
 In[8]: let myseq = iterates succ 1
Out[8]: val myseq : int seq = Cons (1, <fun>)
 In[9]: (=) 1 2
Out[9]: - : bool = false
```

Use = function, partially applied:

```
 In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
 In[11]: filterq ((=) 0) myseq
```

...

(Same examples, but with no new functions . . .)

Adding 1 has a built-in function!

```
  In[7]: succ 1
 Out[7]: - : 2 = int
  In[8]: let myseq = iterates succ 1
 Out[8]: val myseq : int seq = Cons (1, <fun>)
  In[9]: (=) 1 2
 Out[9]: - : bool = false
```

Use = function, partially applied:

```
 In[10]: filterq ((=) 100) myseq
Out[10]: - : int seq = Cons (100, <fun>)
 In[11]: filterq ((=) 0) myseq
          ...
```

```
In[12]: let next a x = (a /. x +. x) /. 2.0

Out[12]: val next : float -> float -> float = <fun>
```

Close enough?

```
In[13]: let rec within eps = function
          | Cons (x, xf) ->
              match xf () with
              | Cons (y, yf) ->
                  if abs_float (x -. y) <= eps then y
                  else within eps (Cons (y, yf))
        Warning 8: this pattern-matching is not exhaustive.
                   ...

Out[13]: val within : float -> float seq -> float = <fun>
```

```
In[12]: let next a x = (a /. x +. x) /. 2.0

Out[12]: val next : float -> float -> float = <fun>
```

Close enough?

```
In[13]: let rec within eps = function
          | Cons (x, xf) ->
              match xf () with
              | Cons (y, yf) ->
                  if abs_float (x -. y) <= eps then y
                  else within eps (Cons (y, yf))
        Warning 8: this pattern-matching is not exhaustive.
                   ...

Out[13]: val within : float -> float seq -> float = <fun>
```

```
In[12]: let next a x = (a /. x +. x) /. 2.0

Out[12]: val next : float -> float -> float = <fun>
```

Close enough?

```
In[13]: let rec within eps = function
          | Cons (x, xf) ->
              match xf () with
              | Cons (y, yf) ->
                  if abs_float (x -. y) <= eps then y
                  else within eps (Cons (y, yf))
        Warning 8: this pattern-matching is not exhaustive.
                   ...

Out[13]: val within : float -> float seq -> float = <fun>
```

```
In[12]: let next a x = (a /. x +. x) /. 2.0

Out[12]: val next : float -> float -> float = <fun>
```

Close enough?

```
In[13]: let rec within eps = function
          | Cons (x, xf) ->
              match xf () with
              | Cons (y, yf) ->
                  if abs_float (x -. y) <= eps then y
                  else within eps (Cons (y, yf))
        Warning 8: this pattern-matching is not exhaustive.
                   ...

Out[13]: val within : float -> float seq -> float = <fun>
```

```
 In[12]: let next a x = (a /. x +. x) /. 2.0

Out[12]: val next : float -> float -> float = <fun>
```

Close enough?

```
 In[13]: let rec within eps = function
           | Cons (x, xf) ->
               match xf () with
               | Cons (y, yf) ->
                   if abs_float (x -. y) <= eps then y
                   else within eps (Cons (y, yf))
         Warning 8: this pattern-matching is not exhaustive.
                    ...

Out[13]: val within : float -> float seq -> float = <fun>
```

```
 In[12]: let next a x = (a /. x +. x) /. 2.0

Out[12]: val next : float -> float -> float = <fun>
```

Close enough?

```
 In[13]: let rec within eps = function
           | Cons (x, xf) ->
               match xf () with
               | Cons (y, yf) ->
                   if abs_float (x -. y) <= eps then y
                   else within eps (Cons (y, yf))
         Warning 8: this pattern-matching is not exhaustive.
                    ...

Out[13]: val within : float -> float seq -> float = <fun>
```

Square roots:

```
In[14]: let root a = within 1e-6 (iterates (next a) 1.0)

Out[14]: val root : float -> float = <fun>

In[15]: root 3.0

Out[15]: - : float = 1.732050807568877719
```

Square roots:

```
In[14]: let root a = within 1e-6 (iterates (next a) 1.0)

Out[14]: val root : float -> float = <fun>

In[15]: root 3.0

Out[15]: - : float = 1.7320508075688772
```

Square roots:

```
 In[14]: let root a = within 1e-6 (iterates (next a) 1.0)
Out[14]: val root : float -> float = <fun>
 In[15]: root 3.0
Out[15]: - : float = 1.732050807568877719
```

Square roots:

```
 In[14]: let root a = within 1e-6 (iterates (next a) 1.0)
Out[14]: val root : float -> float = <fun>
 In[15]: root 3.0
Out[15]: - : float = 1.7320508075688772
```

Square roots:

```
 In[14]: let root a = within 1e-6 (iterates (next a) 1.0)
Out[14]: val root : float -> float = <fun>
 In[15]: root 3.0
Out[15]: - : float = 1.73205080756887719
```

Aside: Newton-Raphson Method

Series is:

$$\begin{aligned} x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} \\ x_2 &= x_1 - \frac{f(x_1)}{f'(x_1)} \\ x_3 &= \vdots \\ x_4 &= \vdots \\ x_5 &= \vdots \end{aligned}$$

We're trying to solve $x = \sqrt{a}$,

i.e. to find a root for $f(x) = x^2 - a$

We have $f'(x) = 2x$ and hence

$$\begin{aligned} x - \frac{f(x)}{f'(x)} &= x - (x^2 - a)/2x \\ &= (x + a/x)/2 \end{aligned}$$

(hence the definition of `next`)